Netzprogrammierung
Weitere Modelle der Netzprogrammierung

Prof. Dr.-Ing. Robert Tolksdorf
Freie Universität Berlin
Institut für Informatik
Netzbasierte Informationssysteme
mailto: tolk@inf.fu-berlin.de
http://www.robert-tolksdorf.de

Überblick

1. Schwächen des RPC-Konzepts
2. Koordinationssprachen
3. Agenten

Schwächen des RPC Konzepts

RPC

- RPC ist
  - sehr populär
  - weit übertragbar
  - implementierbar
- Aber:
  - RPC ist nicht abschließende Lösung
- [Tanenbaum/vanRenesse88] diskutieren einige der Probleme

Konzeptionelle Probleme

- Problem: Rollenidentifikation als Klient oder Server
  - sort < infile | uniq | wc -l > outfile

  - Wer liest, wer schreibt, wer betreibt die Berechnung?
    - fordert wc Zeilen vom uniq Prozess an?
    - fordert uniq den wc Prozess zur Weiterverarbeitung auf?
- Problem: Rollenwechsel in der Interaktion
  - Änderungsbenachrichtigungen an Klienten
    - Klienten sind dann auch Server
  - Signale des Klienten an Server
- Problem: Mehrparteieninteraktionen
  - Datenverteilung an mehrere Server

Weitere Probleme

- Transparenz bei Parametern
  - Zeiger
  - Globale Variablen
- Fehler
  - Server hat Fehler -> Klient blockiert
  - Klient hat Fehler -> Server steht alleine
  - Exactly-Once-Semantik schwer zu etablieren -> I/O
- Nebenläufigkeit
  - Blockierter Klient beim Aufruf (bei synchronem RPC)
  - Partielle Ergebnisse können nicht zur Weiterverarbeitung abgeliefert werden (z.B. bei Datenbankanfrage)

Alternativen zu RPC

- Neben dem RPC Konzept gibt es weitere Versuche, andere Interaktionsmodelle für Netzprogrammierung zu bilden
  - Koordinationssprachen: Tupelraum
  - Peer-to-Peer: Freie Rollen
  - Agenten: Autonome Komponenten

Koordinationssprachen

Koordinationssprachen

- Haben ihren Ursprung in der parallelen Programmierung mit der Sprache Linda
- Sind für netzbasierte Systeme ebenfalls anwendbar
- Haben sich dort zu einer Alternative zu RPC Modellen etabliert
- Sind teilweise kommerziell anerkannt
  - JavaSpaces, Sun
  - TSpaces, IBM

Koordinationssprachen

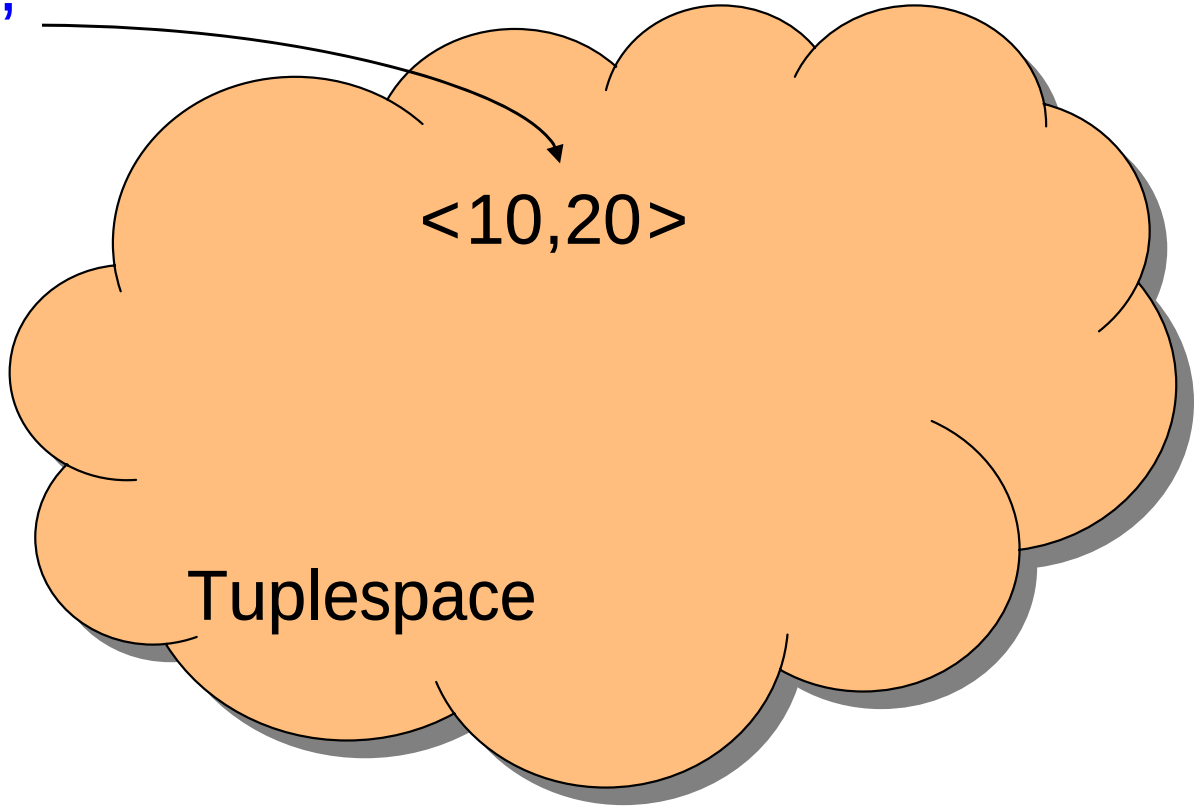
- Haupteigenschaften
  - *Kommunikationspartner sind anonym zueinander*
  - *Lebensdauer der Kommunikationspartner muss nicht überlappen / Kommunikation ist asynchron*

  - Entitäten kommunizieren nur indirekt über einen gemeinsamen Datenraum (Tuplespace)
  - Mehrparteienkommunikation möglich
  - Inhärent nebenläufig
  - Abstrahiert völlig von Orten der Teilnehmer
    -> verteiltes Modell für Netzprogrammierung

- Hauptproblem: Skalierbarkeit

Indirekte Interaktion

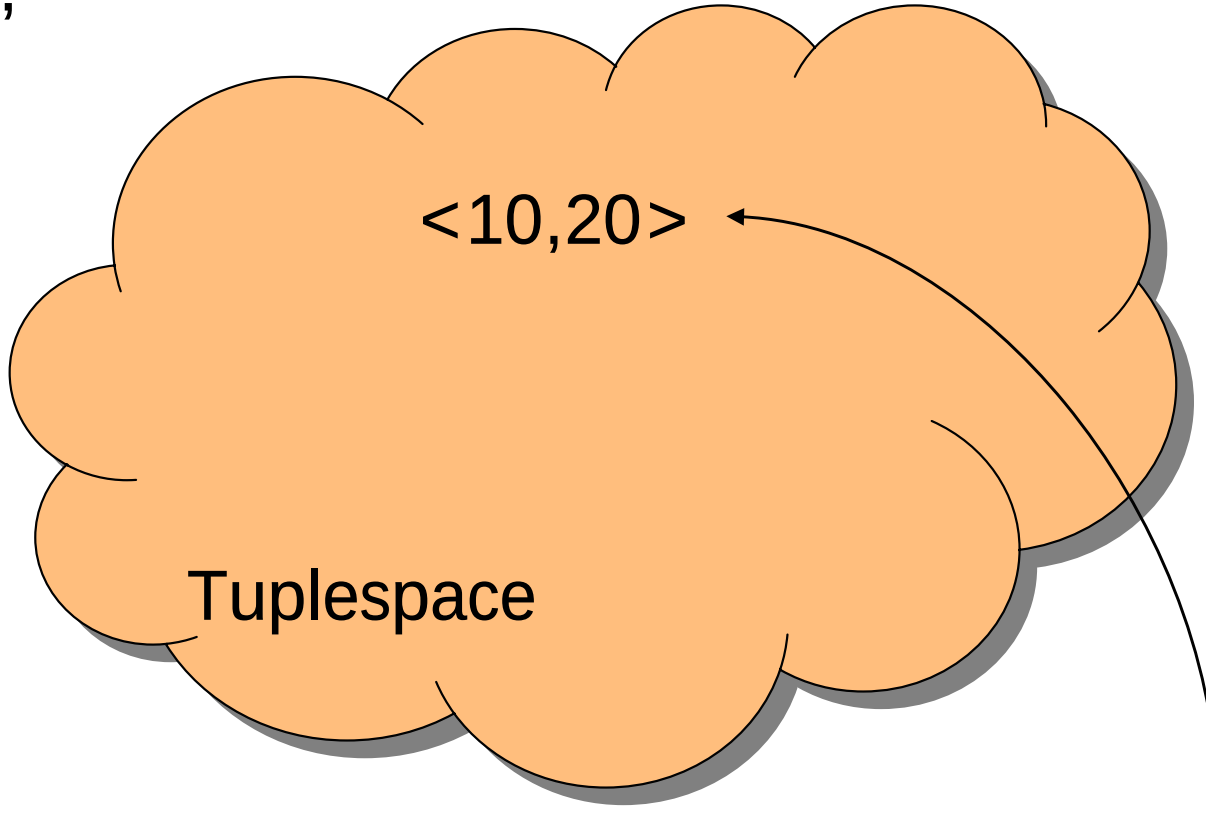
out(10,20);
in(?result);



in(?a,?b);
out(a+b);

Indirekte Interaktion

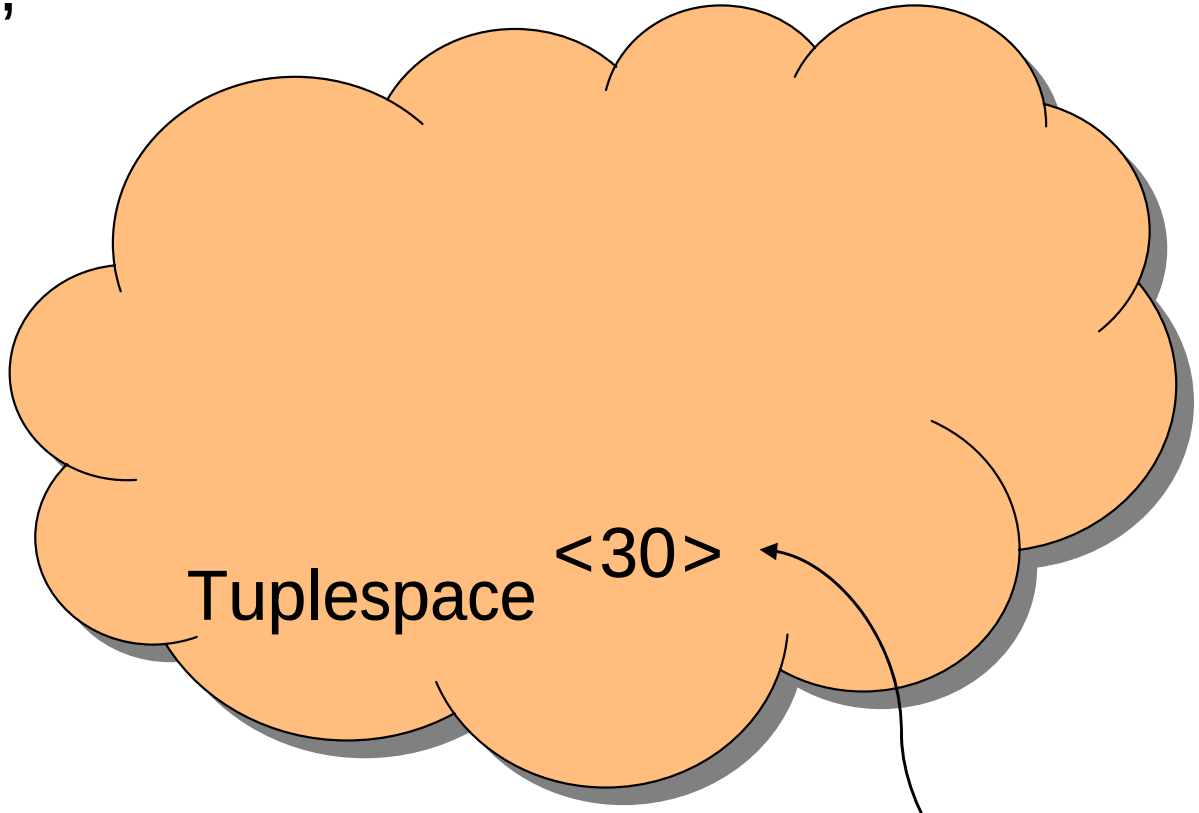
```
out(10,20);
in(?result);
```




```
in(?a,?b);
out(a+b);
```

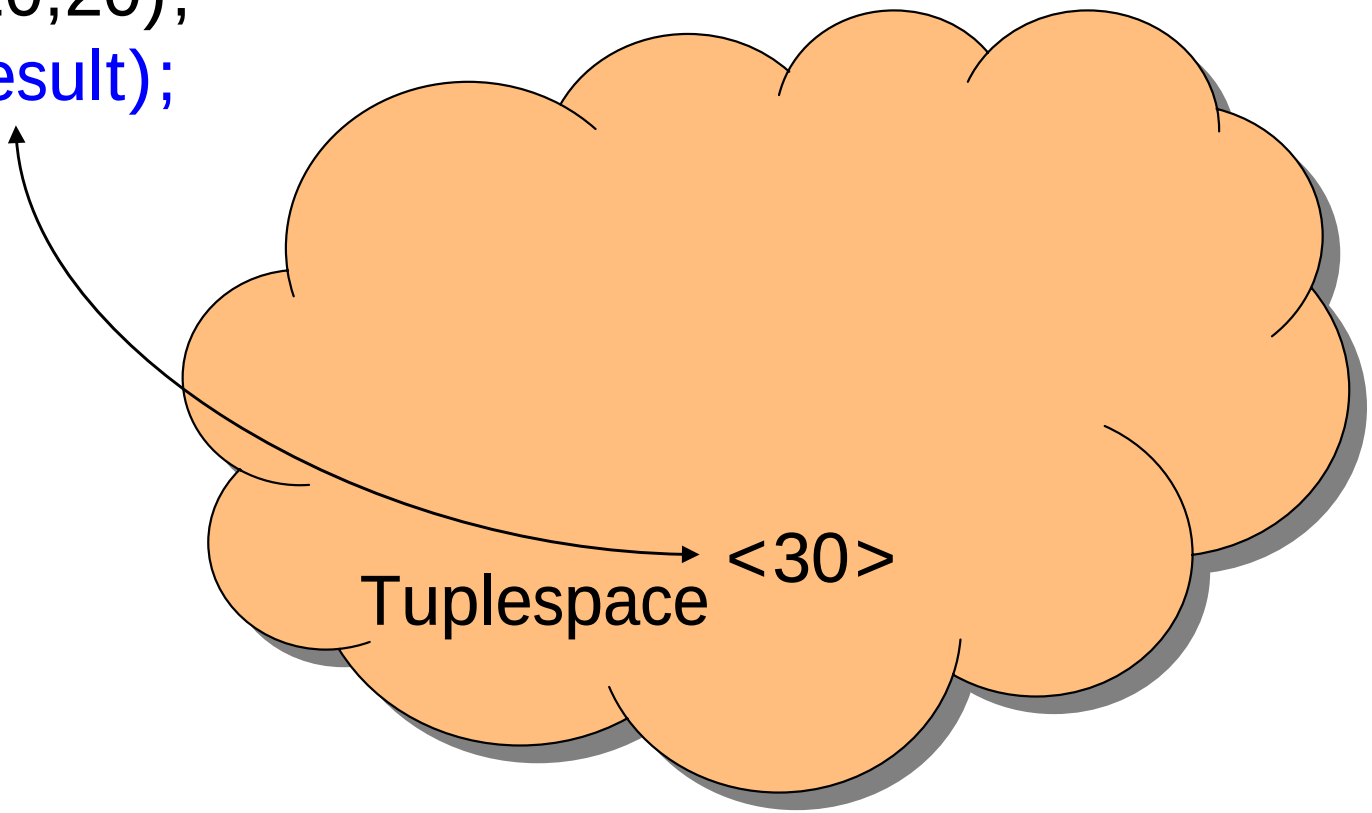
Indirekte Interaktion

out(10,20);
in(?result);



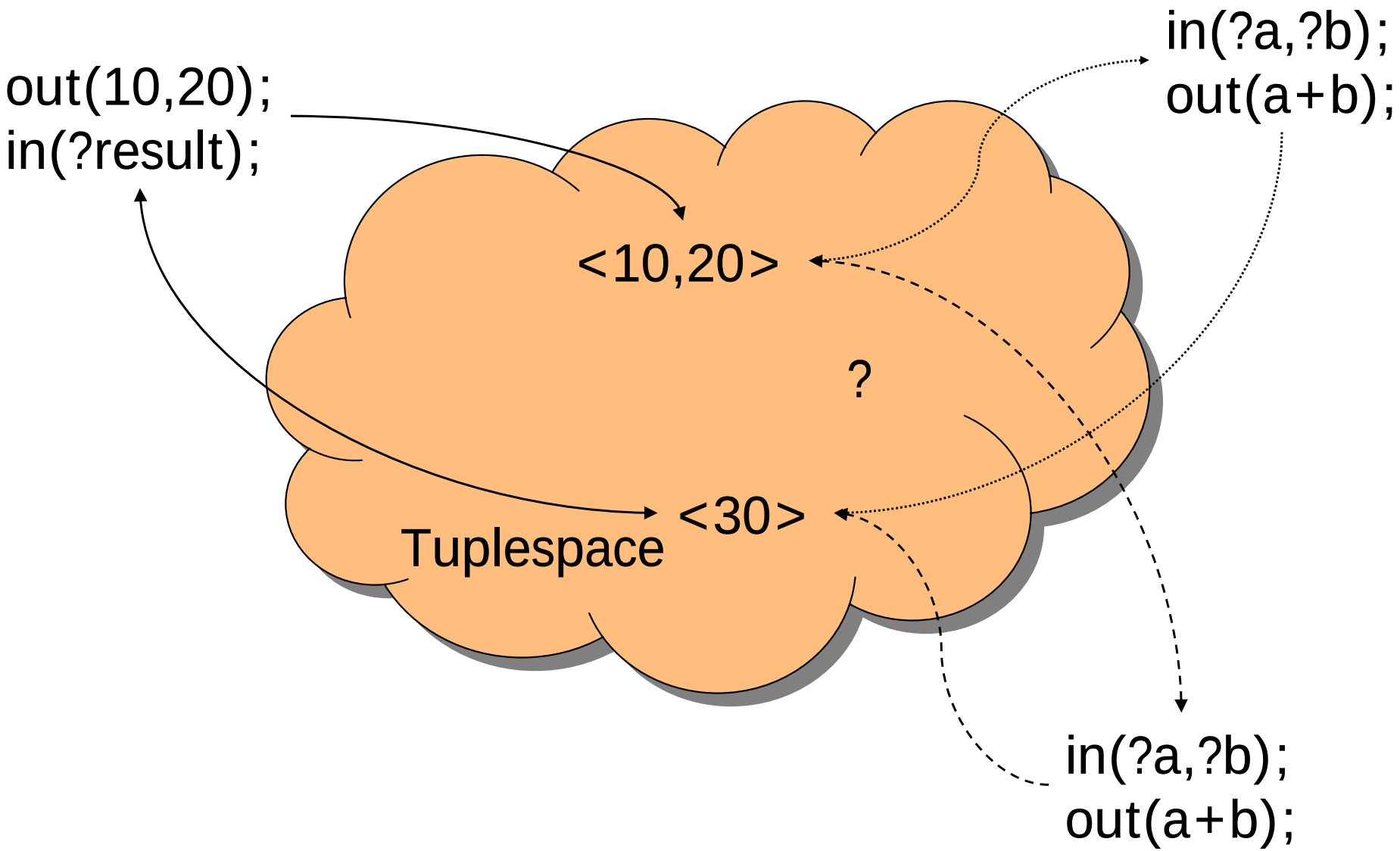
Indirekte Interaktion

out(10,20);
in(?result);

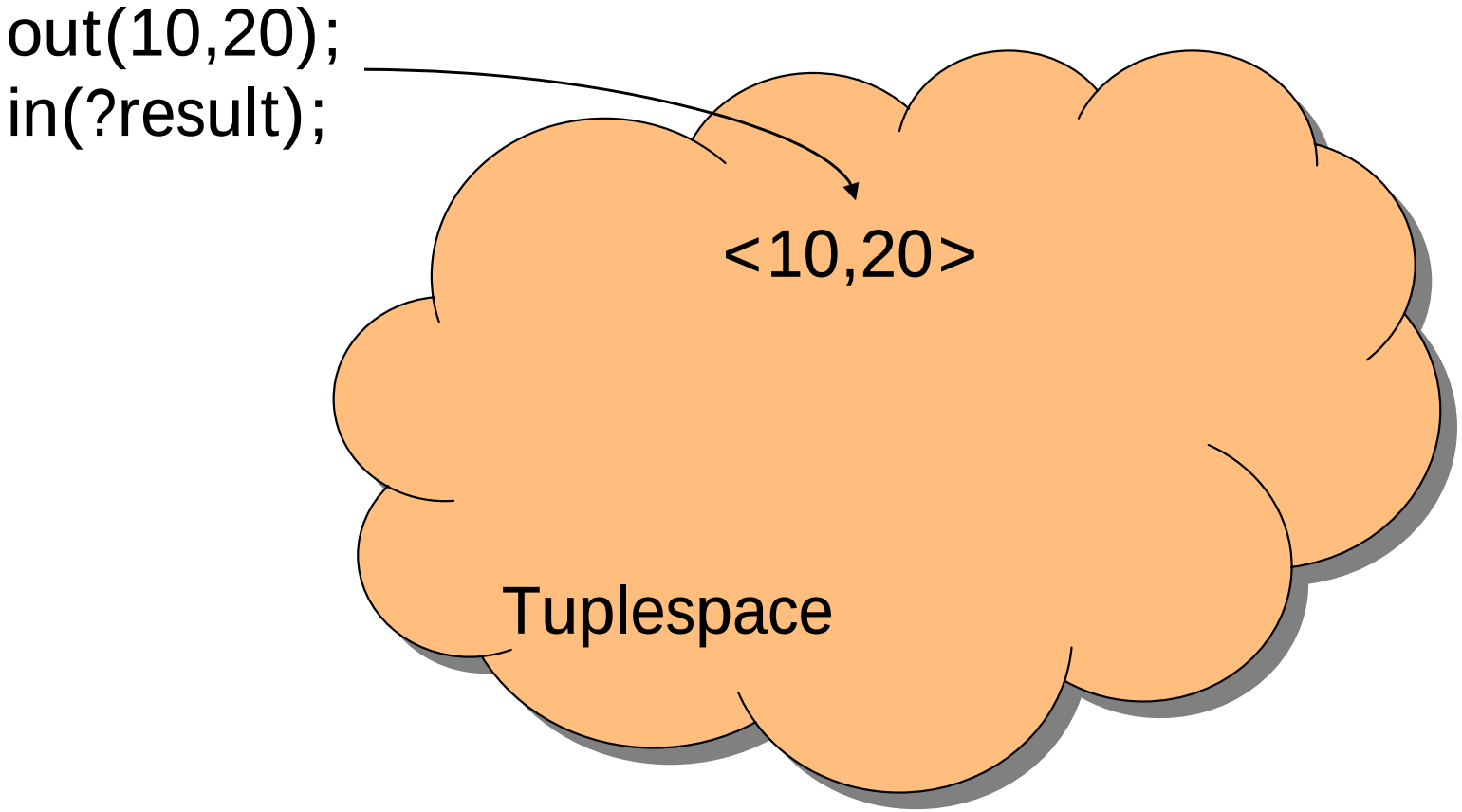


in(?a,?b);
out(a+b);

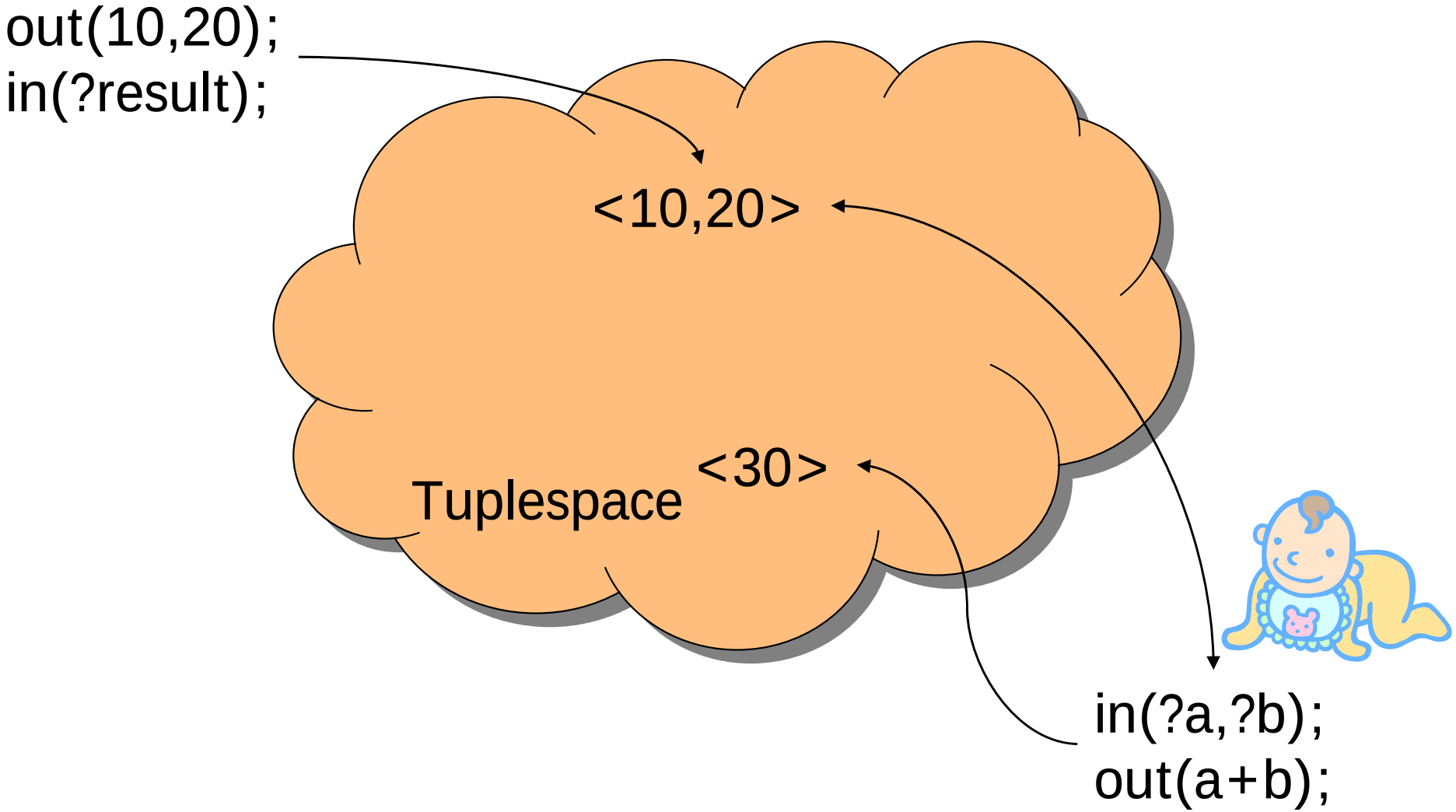
Anonyme Interaktion



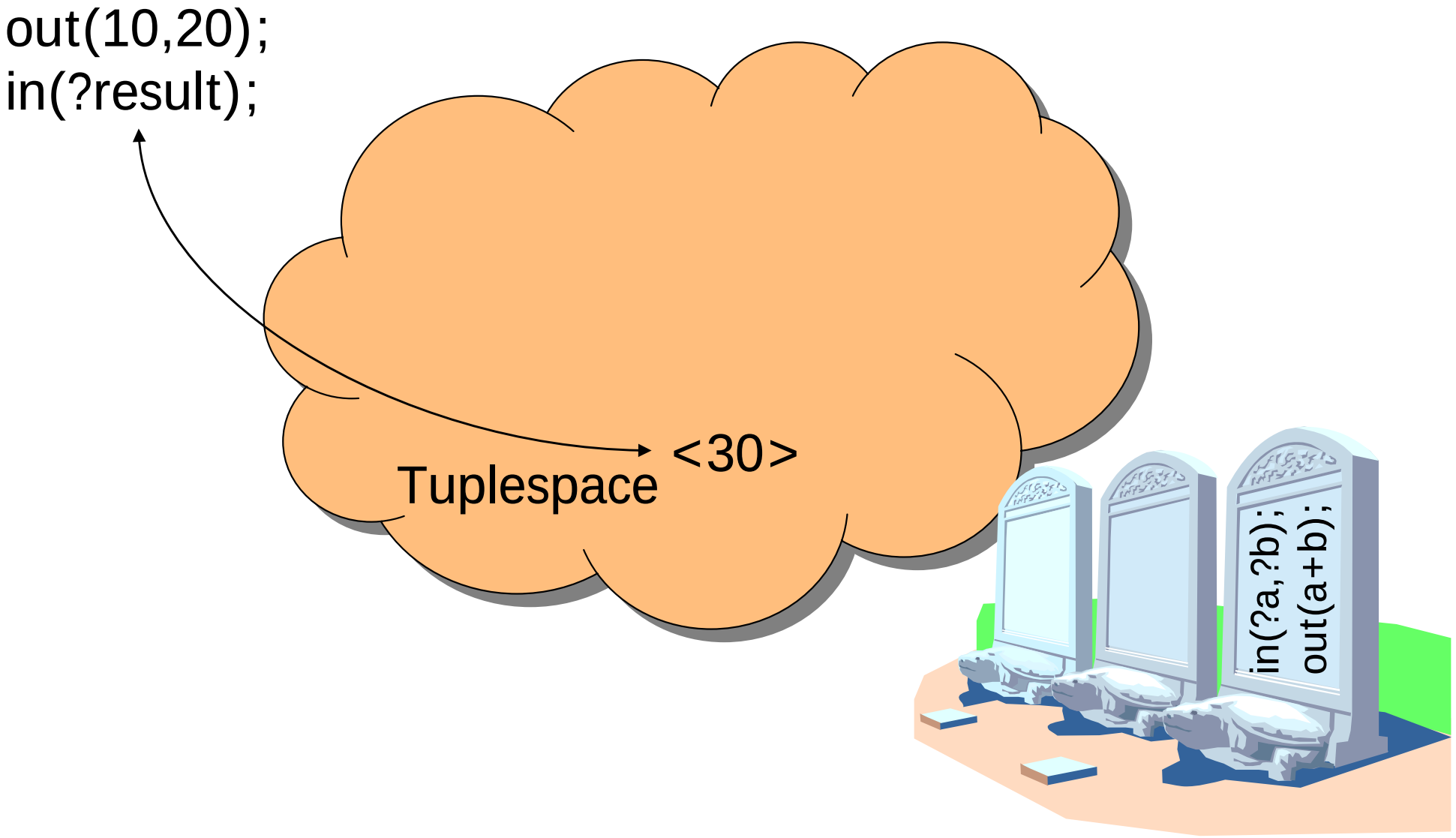
Unterschiedliche Lebensdauer



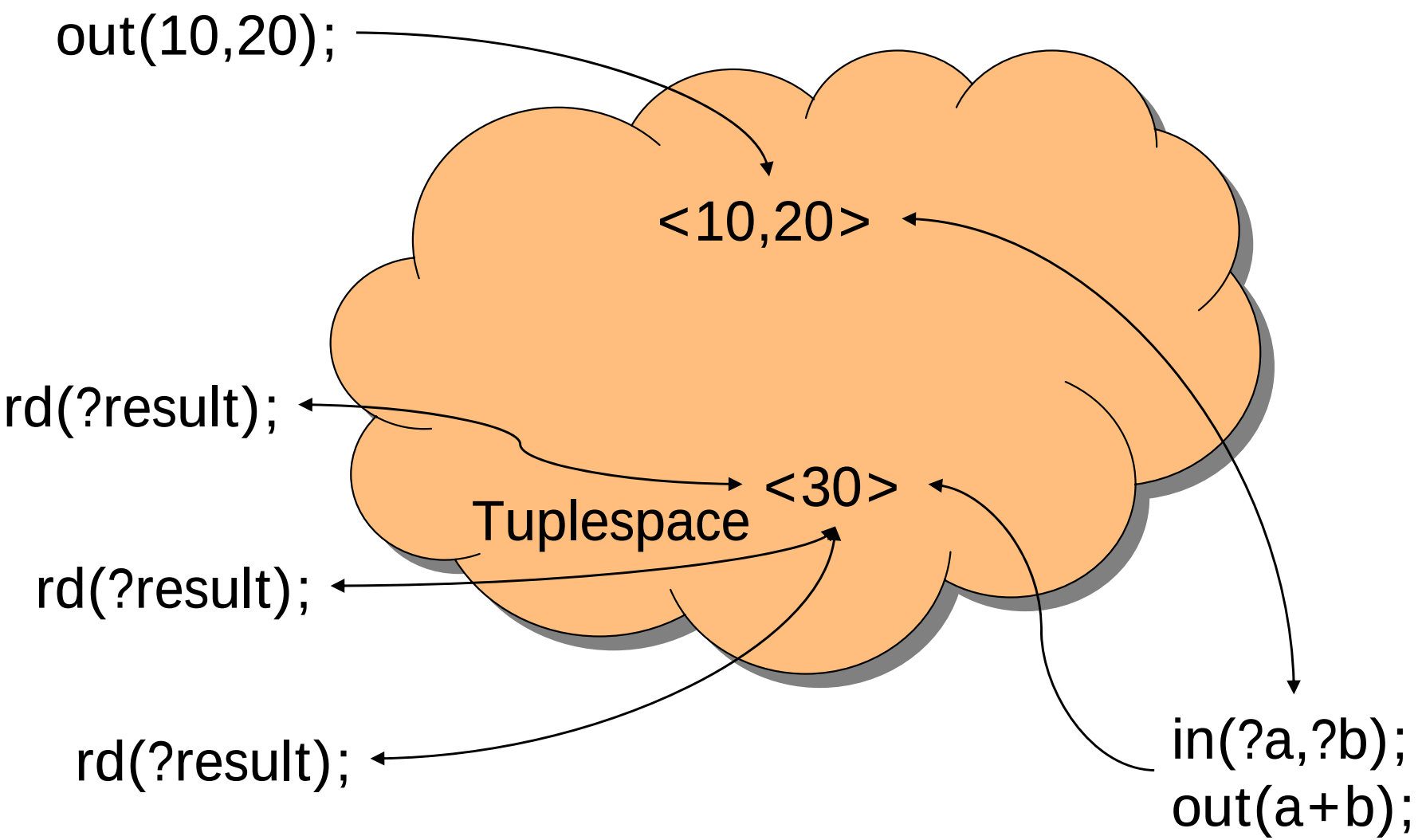
Unterschiedliche Lebensdauer



Unterschiedliche Lebensdauer



Mehrparteieninteraktion



Operationen

- out(tuple): Ablegen eines Tupels in den Tuplespace
- in(template): Herausnehmen eines passenden Tupels aus dem Tuplespace
  - Blockiert bis passendes Tupel vorliegt
    - Gleiche Anzahl Felder
    - Gleiche Typen der Felder
    - Gleicher Wert falls vorhanden
  - Zu <1,2,“start“> passen
    - <?int, ?int, ?string>
    - <1, ?int, ?string>
    - <1,2,“start“>

    aber nicht
    - <?int, ?int>
    - <?int, ?string, ?int>
    - <10, ?int, ?string>
    - <1,2,3>

Operationen

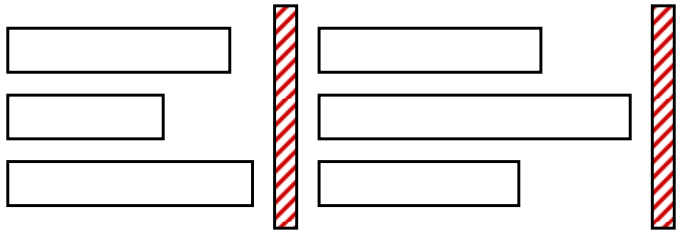
- rd(template):
  Auslesen eines passenden Tupels aus dem Tuplespace
  - Nur Kopie, Tupel bleibt erhalten
  - Konkurrenz mit anderen passenden out

- eval(f()): Evaluation von f() parallel zum laufeden Prozess, Ergebnis als Tuple in Tuplespace
  - P:eval(f());q... ähnlich (out(f()))||q...

- Etwas von einem anderen Prozess ausführen lassen:
  - Worker/“Server“:
    - in("Rechendienst",?p1,?p2);
      r=f(p1,p2);
      out("RechendienstErg",r);
  - „Client“
    - out("Rechendienst",10,30);
      in("RechendienstErg",?r);
- Es kann beliebig viele Worker geben:
  Bag-of-tasks - replicated-worker Muster
  - Aufgabe wird an einen Worker nichtdeterministisch vergeben
  - System kann Last ausgleichen
  - Bei mehreren „Klienten“ muss Auftrag durch ID unterschieden werden

- *Verteilte* Semaphore
  - V-Operation: out("sem")
  - P-Operation: in ("sem")
  - Initialisierung: out("sem") n Mal wiederholen

- Barrier Synchroniziation (Schrankensynchronisation)



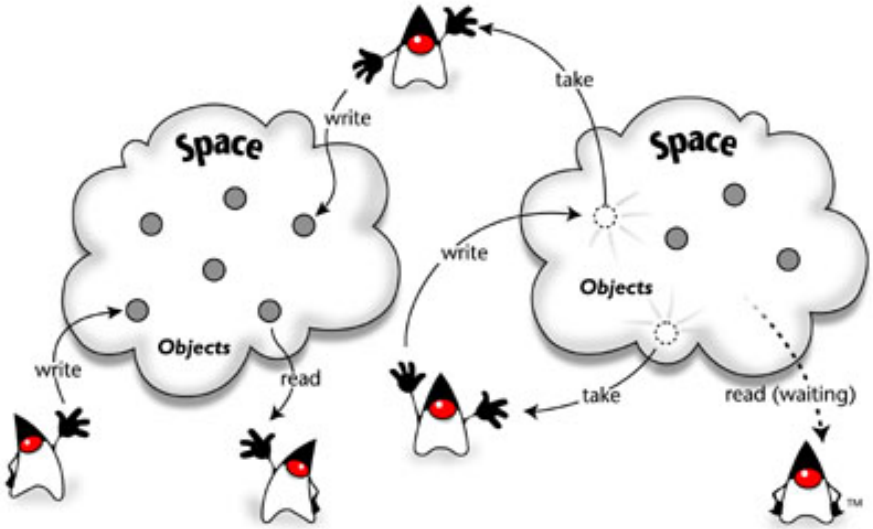
- Schranke b1 erzeugen auf die 3 Prozesse warten sollen
  - out("b1",3)
- Jeder Prozess dekrementiert und wartet auf das „Eintreffen“ aller Prozesse:
  - in("b1",?v);
    out("b1",?v-1);
    rd("b1",0);

- Verteilte Schliefe
  - for (loop control)
    (something)
  - Wenn eval Prozesse auf unterschiedlichen Maschinen platziert:
    - for (loop control)
      eval("loop", something());
      for (loop control)
      in ("loop", 1);
    - something() endet mit out("loop", 1);

Probleme

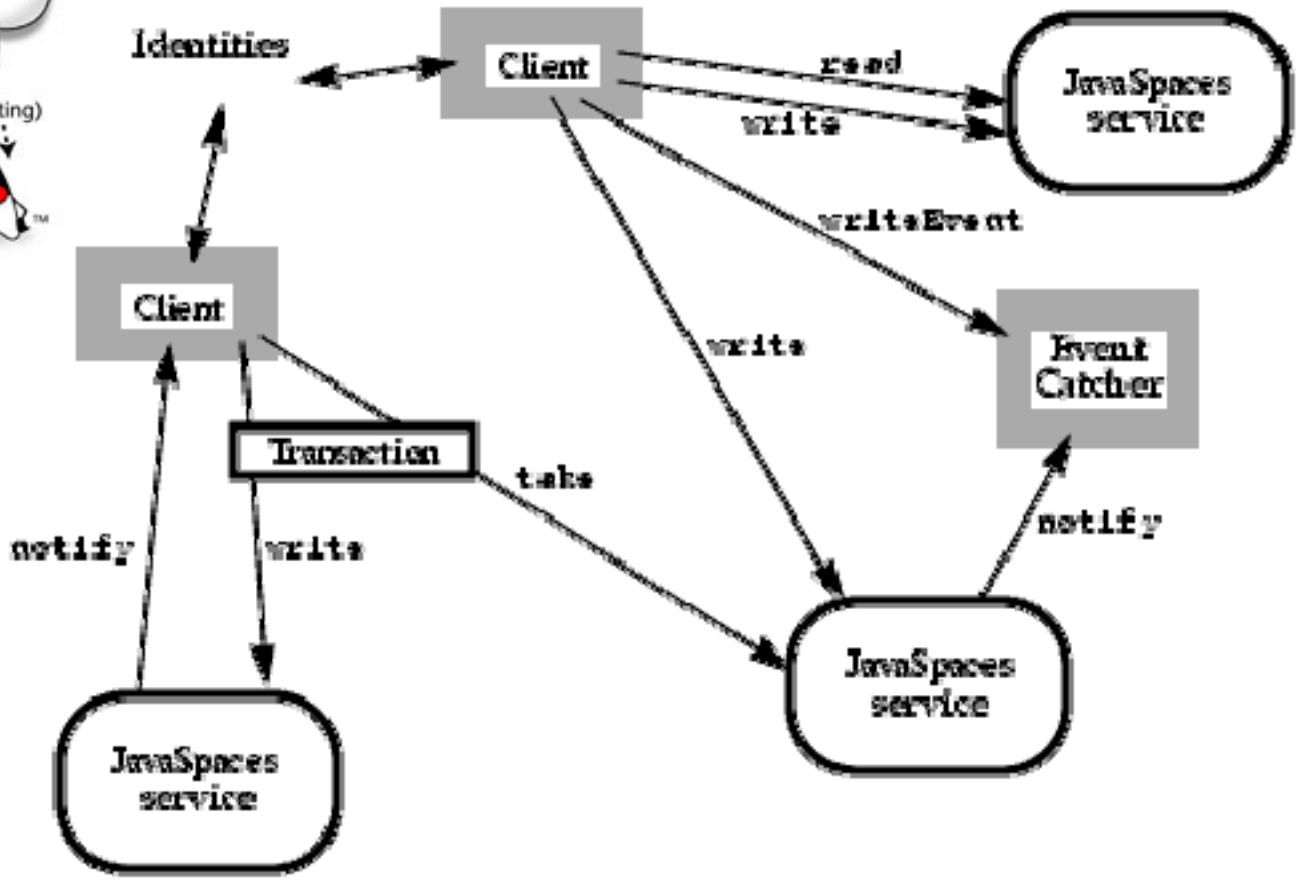
- Fehlertransparenz
  - Was passiert bei Fehler zwischen in(?a,?b); und out(a+b);?
- Termination von Interaktionen
  - Was passiert mit dem <30> Tupel, wenn alle interessierten Prozesse rd(?result); gemacht haben?
- Skalierbarkeit
  - Wie verteilt man den Tuplespace effizient?

JavaSpaces



- write
- take
- read
- Benachrichtigungen
- Transaktionen

Implementierung von Linda von Sun



Beispiel Entry Objekt

```
package jsbook.chapter1.helloWorldTwo;
import net.jini.core.entry.Entry;
public class Message implements Entry {
    public String content;
    public Integer counter;
    public Message() {  }
    public Message(String content, int initVal) {
        this.content = content;
        counter = new Integer(initVal);
    }
    public String toString() {
        return content + " read " + counter + " times.";
    }
    public void increment() {
        counter = new Integer(counter.intValue() + 1);
    }
}
```

Beispiel

Freie Universität Berlin

```
package jsbook.chapter1.helloWorldTwo;
import jsbook.util.SpaceAccessor;
import net.jini.core.lease.Lease;
import net.jini.space.JavaSpace;
public class HelloWorldClient {
    public static void main(String[] args) {
        try {
            JavaSpace space = SpaceAccessor.getSpace();
            Message template = new Message();
            for (;;) {
                Message result = (Message)
                    space.take(template, null, Long.MAX_VALUE);
                result.increment();
                space.write(result, null, Lease.FOREVER);
                Thread.sleep(1000);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Beispiel

```
package jsbook.chapter1.helloWorldTwo;
import jsbook.util.SpaceAccessor;
import net.jini.core.lease.Lease;
import net.jini.space.JavaSpace;
public class HelloWorld {
    public static void main(String[] args) {
        try {
            Message msg = new Message("Hello World", 0);
            JavaSpace space = SpaceAccessor.getSpace();
            space.write(msg, null, Lease.FOREVER);
            Message template = new Message();
            for (;;) {
                Message result = (Message)
                    space.read(template, null, Long.MAX_VALUE);
                System.out.println(result);
                Thread.sleep(1000);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Freie Universität Berlin

Coordinating Web-based Systems with Documents in XMLSpaces

Beiträge von Dirk Glaubitz

Motivation: Coordination and the Web

Freie Universität Berlin

- Web has become *the* universal information system
- Not many distributed applications utilizing the Web for universal access

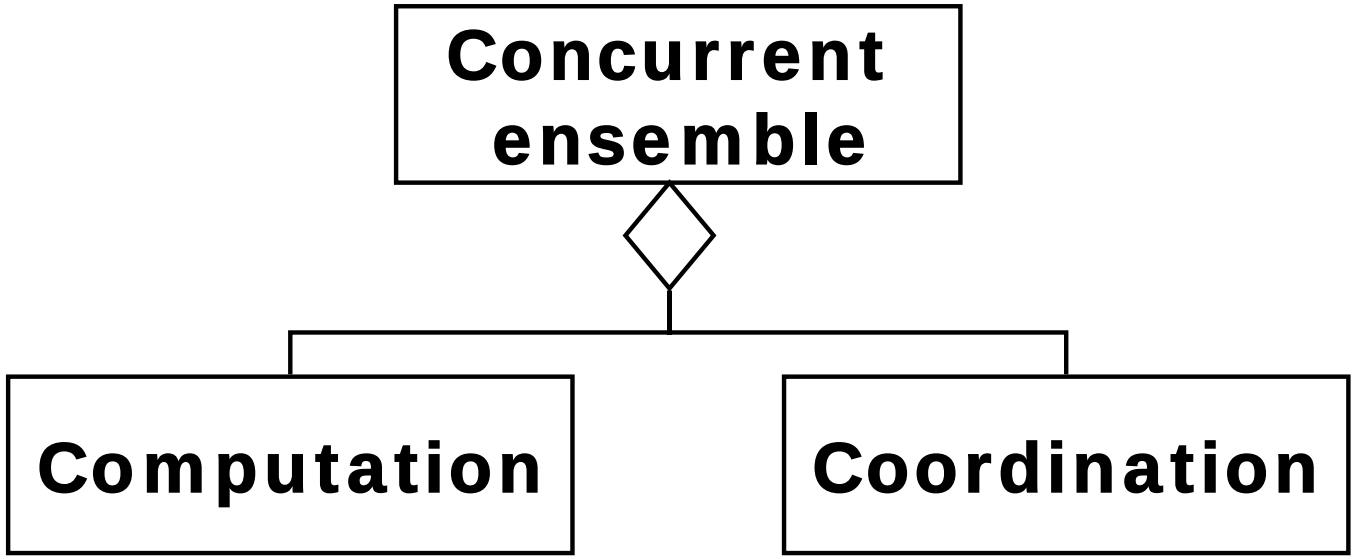
- Core question: What is the concept applied for the coordination of independent activities in a cooperative whole?

- XMLSpaces follows idea of separate coordination language that deals exclusively with the aspects of the interplay of entities and provides concepts orthogonal to computation.

- Coordination theory [Malone, others]:

  Coordination is the management of dependencies between activities

- Coordination technology [Gelernter/Carriero]:



Linda

- Coordination language with tuplespace accessed with few operations: out, in/rd, inp/rdp, eval

- **out(<2000, "Eilat">)**
  out(<2001, "Trento">)
- **in(<2001,?town)**
  Matching relation determines result
- **in(<2002,?town)**
  Blocks until *someone* out´s a match *sometimes*

- Linda is good for Web based systems:
  Uncoupled in space and time, asynchronous, anonymous...

Motivation: Coordination and the Web

Freie Universität Berlin

- Tupels are weak in expressibilty
  - Fixed typing
  - No higher order structures
  - „small“
- Extensible Markup Language XML has become the format to exchange data markup following application specific syntaxis
- XMLSpaces =
  common communication format XML +
  coordination language Linda

Motivation: Coordination and the Web

Freie Universität Berlin

- Features:
  - XML documents serve as field-data
  - Multitude of relations amongst XML docs for matching
  - XMLSpaces is distributed, servers at different locations form one logic dataspace.
  - Distributed events supported so that clients are notified when a tuple is added or removed somewhere

XML documents serve as field-data

```
<?xml version="1.0"?>
<!DOCTYPE address SYSTEM
                "address.dtd">
<location>
  <city name="Trento"/>
  <host>FACOLTÀ DI ECONOMIA</host>
  <street name="Via Vigilio Inama"
          no="5"/>
</location>
```

- <2001, __>

How to match?

- A formal is an object describing an XML document
- Can be:
  - Another document:

```
<?xml version="1.0"?>
<!DOCTYPE address SYSTEM
                  "address.dtd">
<location>
  <city name="Trento"/>
  <host>FACOLTÀ DI ECONOMIA</host>
  <street name="Via Vigilio Inama"
          no="5"/>
</location>
```

in(2001, ?__)

  - Something else
    in(2001, DOCTYPE="address.dtd")

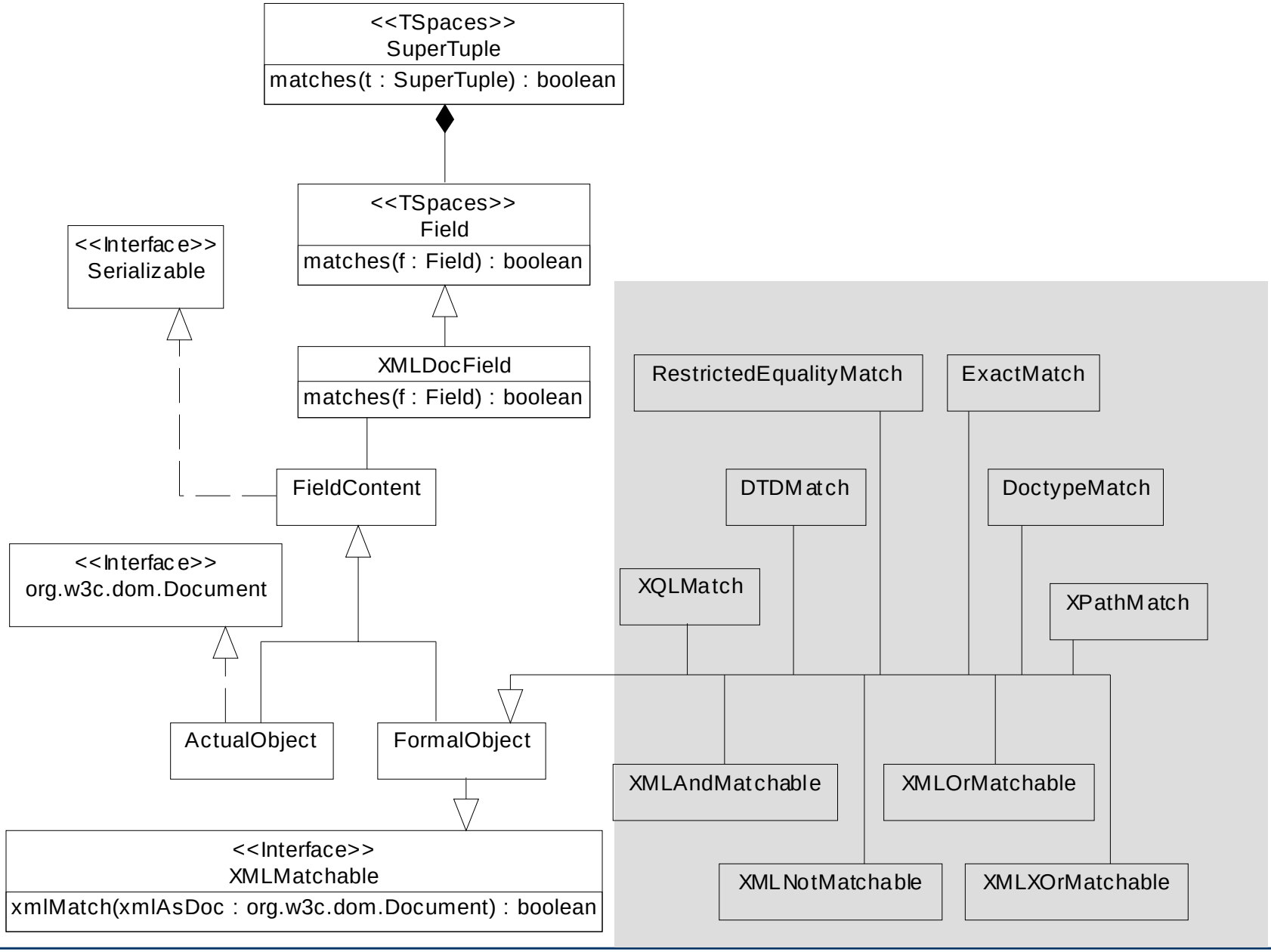
- There is a variety of relations that identify XML documents

Multiple matching relations in XMLSpaces

Freie Universität Berlin

Relation	*Meaning*	*Tool used*
Exact equality	Exact textual equality	DOM interfaces
Restricted equality	Textual equality ignoring comments, PIs, etc.	DOM interfaces
DTD	Valid towards a DTD	XML4J
DOCTYPE	Uses specific Doctype name	DOM
XPath	Fulfills an XPath expression	Xalan-Java
XQL	Fulfills an XQL expression	GMD-IPSI XQL
AND	Fulfills two matching relations	–
NOT	Does not fulfill matching relation	–
OR	Fulfills one or two matching relations	–
XOR	Fulfills one matching relation	–

Class hierarchy for XML tuplefields



Openness

- Further relations can be added easily
- Example from XQL integration:

```
package matchingrelation;
import xmlspaces.XMLMatchable;
import java.io.*;
import org.w3c.dom.Document;
import de.gmd.ipsi.xql.*;
public class XQLMatch implements XMLMatchable{
  String query;
  public XQLMatch(String xqlQuery){
    query = xqlQuery;
  }
  public boolean xmlMatch(Document xmlAsDoc){
    return XQL.match(query, xmlAsDoc);
  }
}
```

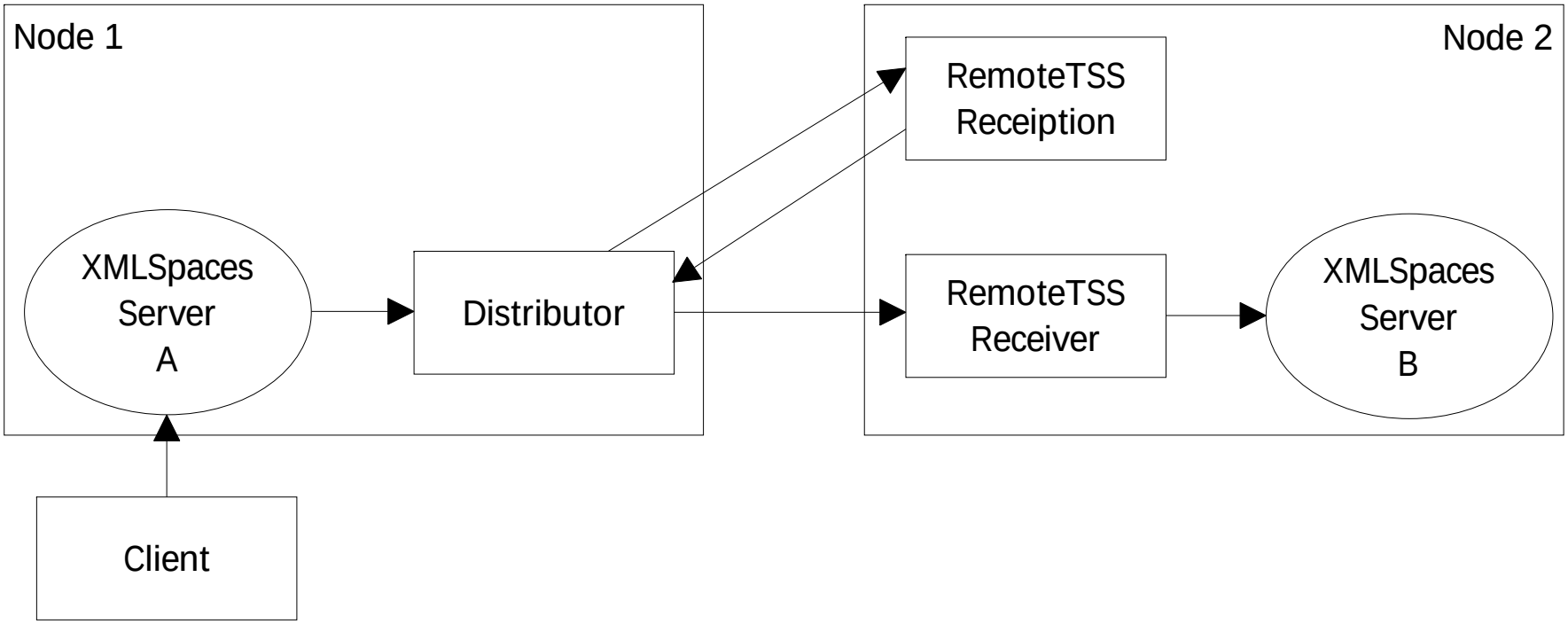
Distribution

- Options known from work on Linda:
  - *Centralized*: One server holds the complete dataspace.
  - *Distributed*: All servers hold distinct subsets of the complete dataspace
  - *Full replication*: All servers hold consistent copies of the complete dataspace
  - *Partial replication*: Subsets of servers hold consistent copies of subsets of the dataspace
  - *Hashing*: Matching tuples and templates are stored at the same server selected by some hashing function

- XMLSpaces supports all schemas and is extensible

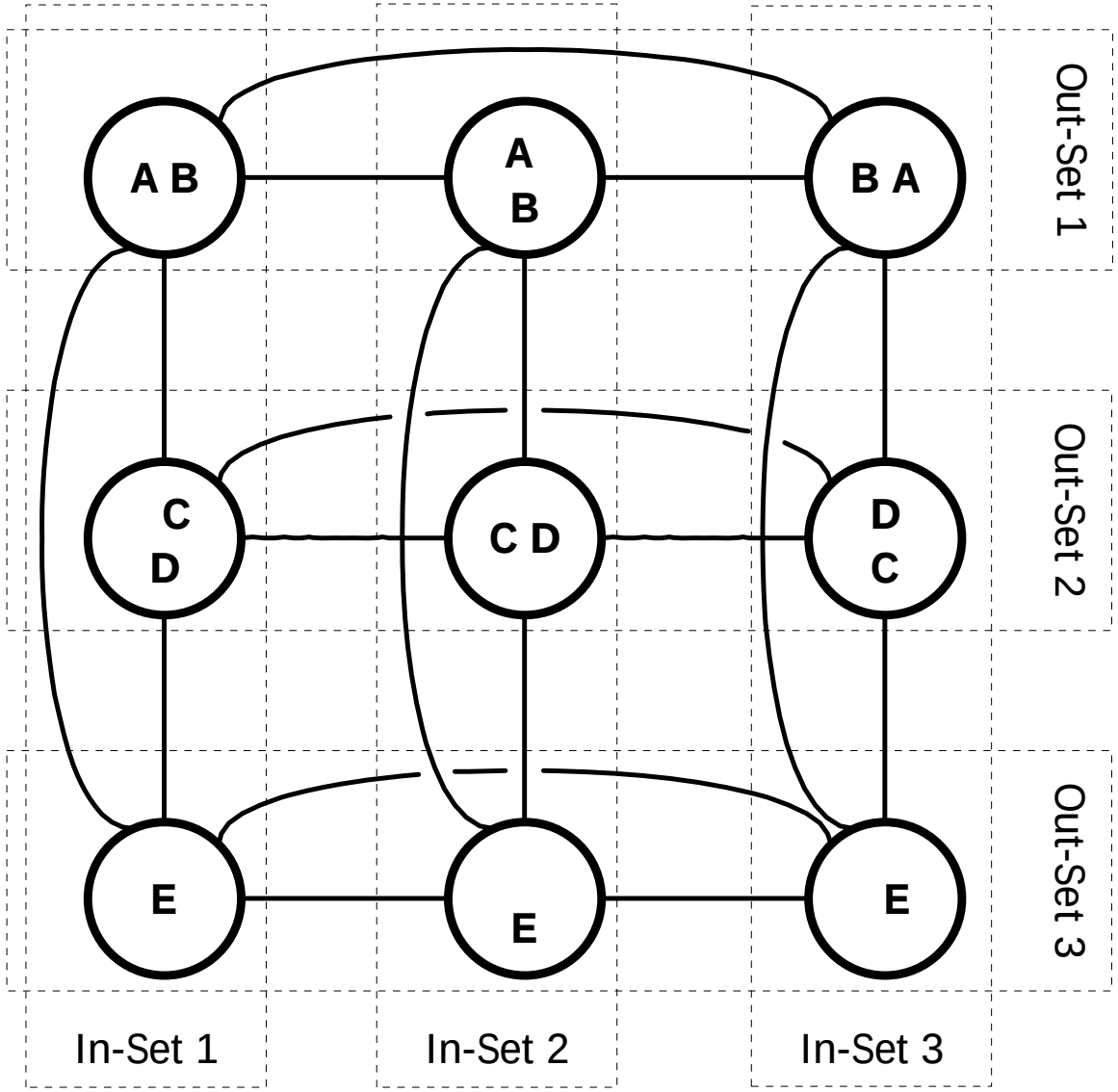
Distributed servers

- Receiption tells node which peers to use
- Each peer is initially contacted by a RemoteTSSReceiption
- Further communication is by a RemoteTSSReceiver
- Distributor object encapsulates distribution schema



Partial replication scheme

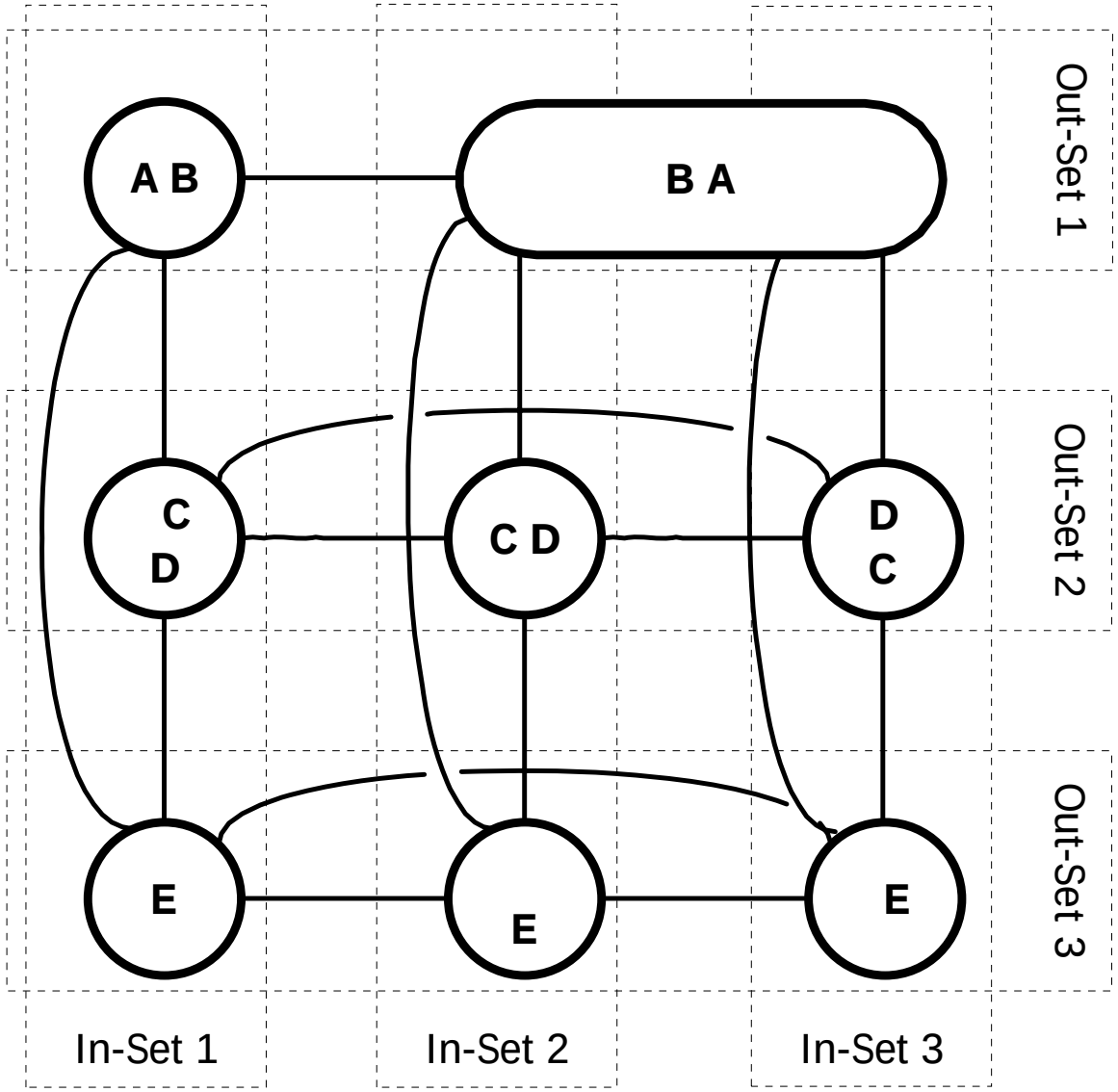
- Nodes in out-set contain replica
- Union of contents in nodes in in-set is whole content of space



Partial replication scheme

- Simulated nodes to keep grid
- Protocols for join and leave of nodes

- Centralized, distributed and fully replicated schemes are special cases of partial replication

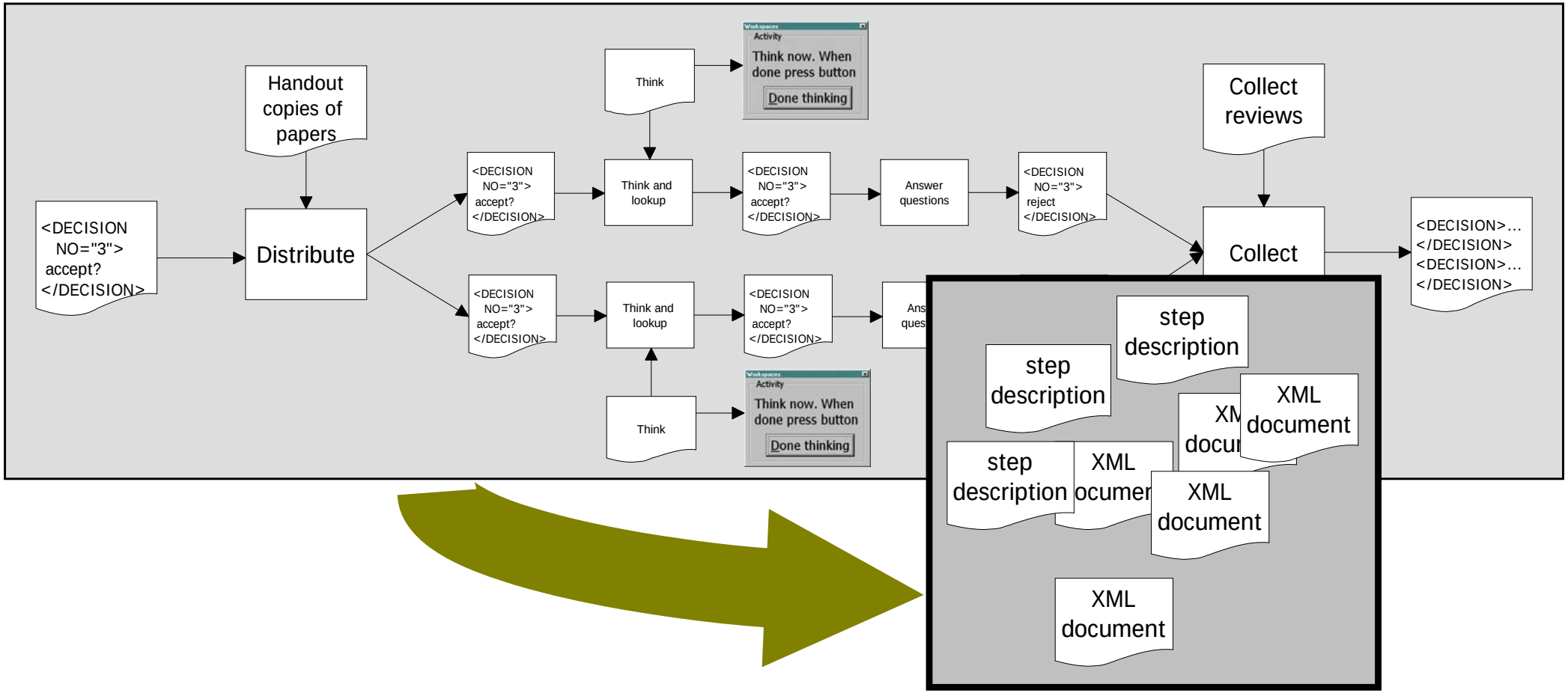


Implementation

- DOM level 1 as internal representation
- TSpaces for local spaces (source licence granted by IBM Almaden)
- Various XML software
- JVM based

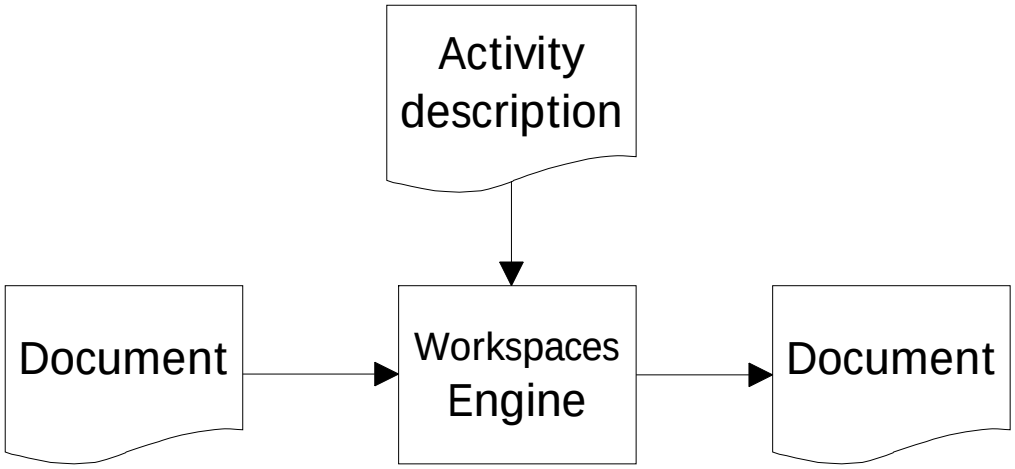
Application: Workspaces

- An XML/XSL based workflow system
- Workflow graph is split into single steps that are represented as XSL documents

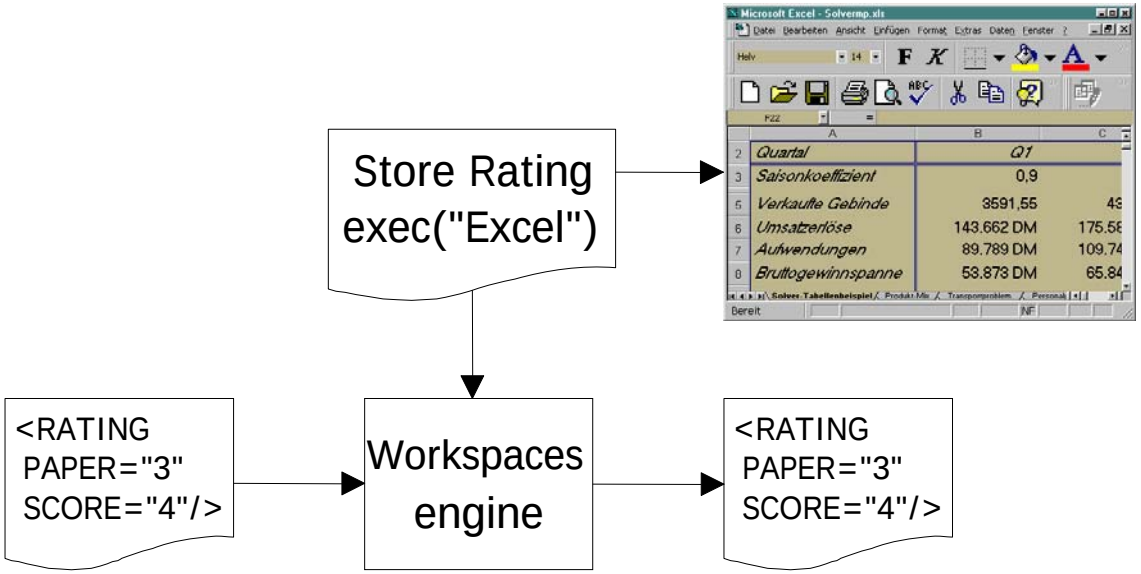


Application: Workspaces

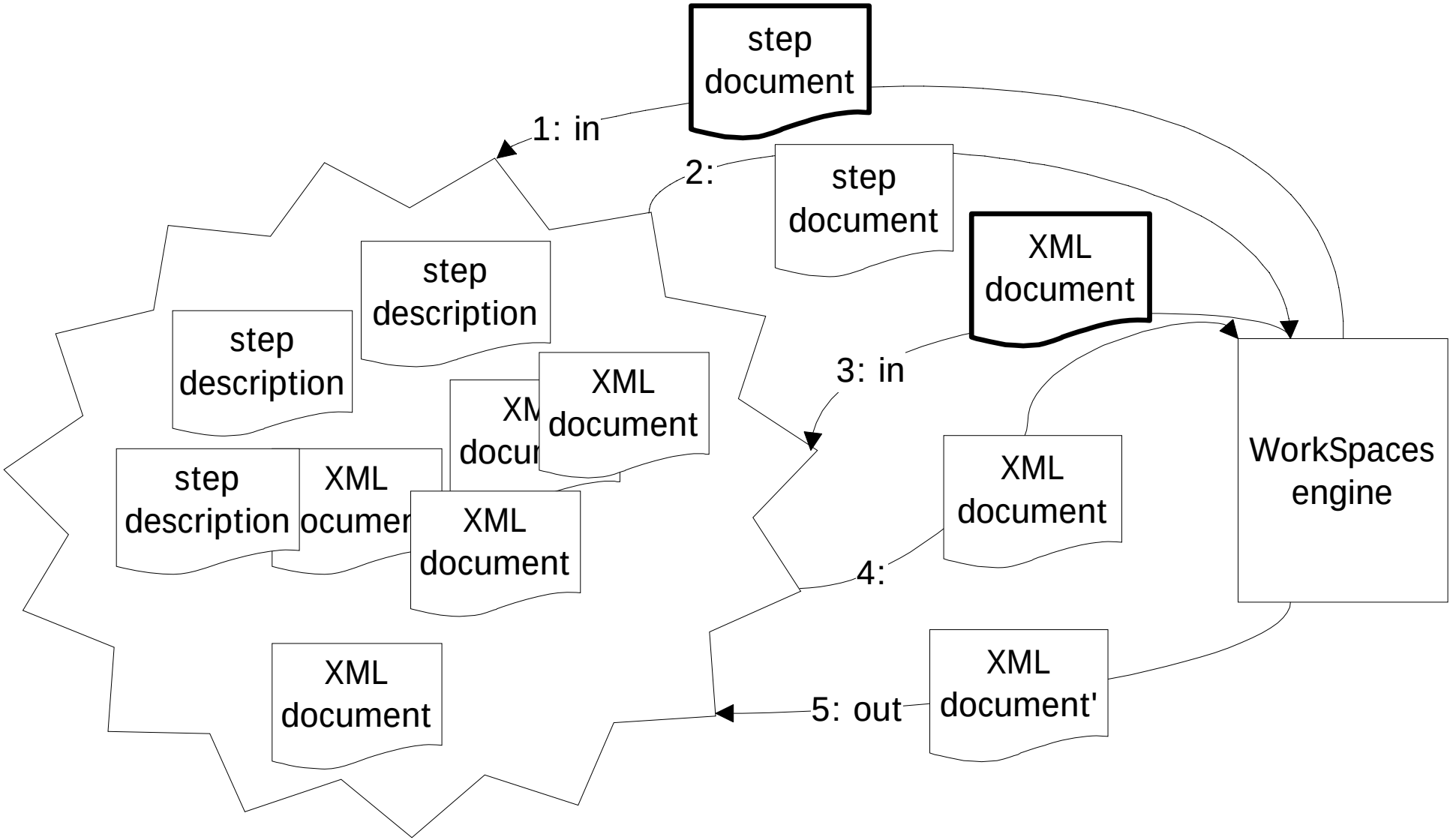
- Documents and worksteps as XML documents



- Workspaces engine is an extended XSL engine



XML access in Workspaces



Conclusion and outlook

- Using XML as tuple-fields adds necessary structure for data
- Relations on XML documents are manifold
- XMLSpaces =
  common communication format XML
  \+ coordination language Linda
- \+ open set of matching relations
- \+ open set of distribution strategies

Agenten

Autonomie

- Die RPC-basierte Marssonde
  - Flugdatenübermittlung per RPC von der Erde
  - Ändert Kurs auf RPC Aufruf
  - Entscheidungen über Kursänderungen werden auf der Erde getroffen
- Unpraktikabel
  - Sehr lange Latenz
  - Entscheidungsfristen kleiner als Netzlatenz
  - Risiko der Fehlersemantik... Realer Absturz...
- Problem
  - Sonde ist nicht autonom in ihren Entscheidungen

Agenten

- Agent ist
  - ein Rechnersystem, das *eigenständig* für einen Benutzer agiert
- Mehragentensystem ist
  - ein Rechnersystem, das aus mehrere Agenten besteht, die miteinander *interagieren*
- Agenten müssen
  - kooperieren
  - sich koordinieren
  - verhandeln

Agenten

- Vorteile
  - Autonomie
  - Verschiebung der Sichtweise auf Systeme von der Berechnung zur Interaktion
  - Besseres Abbild realer Gesellschaften

- Nachteile
  - Unklare Realisierung
  - Unterscheidung zu
    - Verteilten Systemen (aber: Autonomie)
    - KI (aber: Verteiltheit und Realisierungsfrage)
    - Spieltheorie/Ökonomie (aber: Automatisierung)
    - Gesellschaftswissenschaften (aber: Informatiker)

Agenten

- Schwache Definition
  - Autonomie: Agent operiert ohne direkte Steuerung von außen und kontrolliert seine Aktionen selber
  - Responsiveness/Empfindlichkeit: Agent beobachtet Umgebung und reagiert auf Änderungen darin
  - Proaktiveness/Initiativ: Agent reagiert nicht nur, sondern wird selbständig zur Erreichung von Zielen aktiv
  - Sozial: Agent interagiert mit anderen zur Erreichung eigener und deren Ziele
- In der Regel mit Fokus auf Software-Agenten

Agenten

- Starke Definition
  - Mobilität: Agenten bewegen sich in ienem elektronischen Netzwerk (siehe auch: Roboter)
  - Veracity/Aufrichtigkeit: Agenten geben nicht wissentlich falsche Informationen weiter
  - Rationalität: Agenten beschädigen nicht durch Aktionen ihre eigenen Ziele
  - Kooperativität: Agenten arbeiten mit (menschlichen) Auftraggebern zusammen und übernehmen deren Ziele
- Zieht Aspekte menschlichen Verhaltens ein
- ... Intelligent Agents, Smart Agents ...

Architekturen

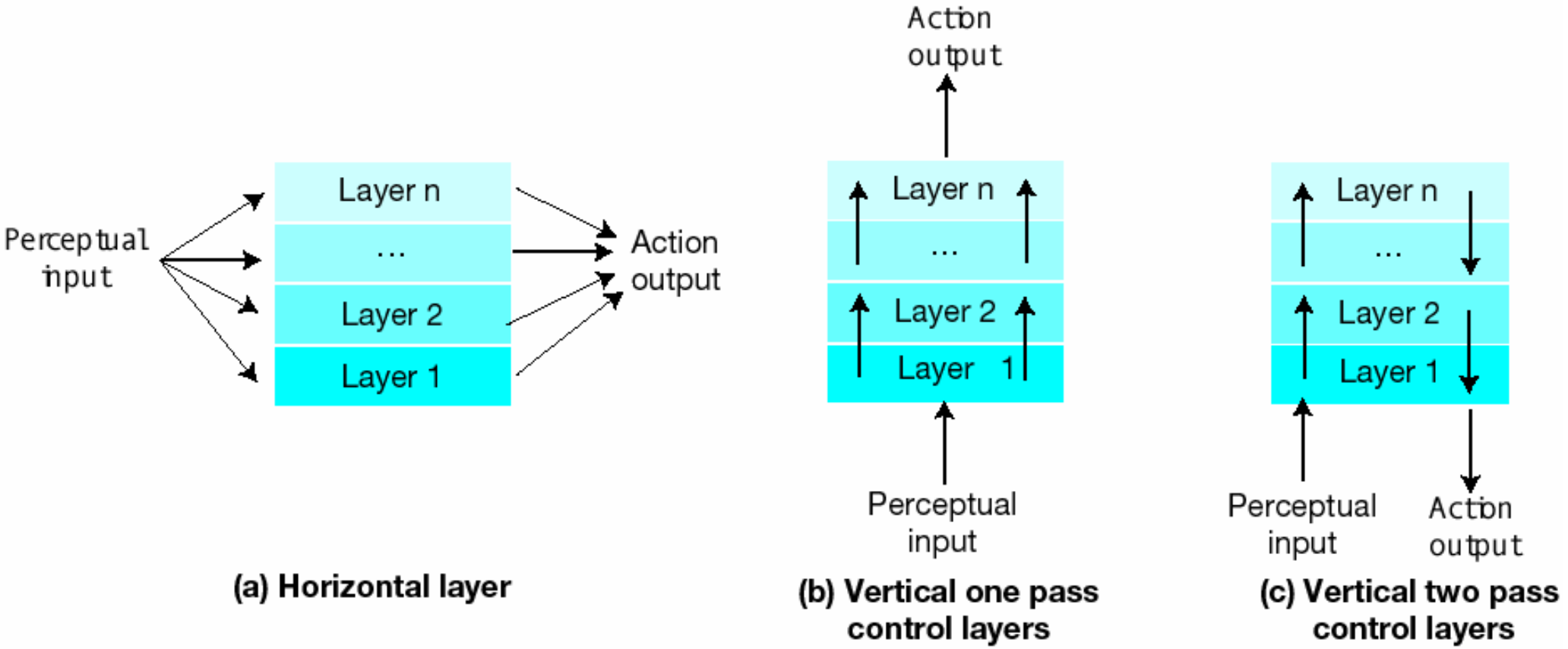
- *Deliberative* (Abwägend) *Architectures*
- Annahme: Intelligente Aktivität durch
  - symbolische Muster zur Repräsentation von Probleme
  - Operationen darauf zur Lösungsgenerierung
  - Suche auf Mustern zur Auswahl von Lösungen
- Agent sollte symbolisches Modell seiner Welt haben
- Beliefs, Desire, Intentions (BDI) Agenten:
  - Jeweils Modelle explizit repräsentiert
  - Tief erforscht
  - Schlecht skalierbar

Architekturen

- *Reactive Architectures*
- Annahme: Intelligentes Verhalten durch Beobachung der Umgebung und Reaktion darauf
- Ohne interne Repräsentation
- Entscheidungen getroffen
  - mit wenig Informationen
  - einfachen Regeln (situativ)
  - in Echtzeit
- -> Autonome Roboter

- *Hybrid Architectures*
  - Kombination aus Deliberative/Reactive Architectures

- *Layered (geschichtete) Architectures*
- Agent besteht aus Subsysteme, die Problemteile verarbeiten
- Unterschiedliche Konfigurationen:



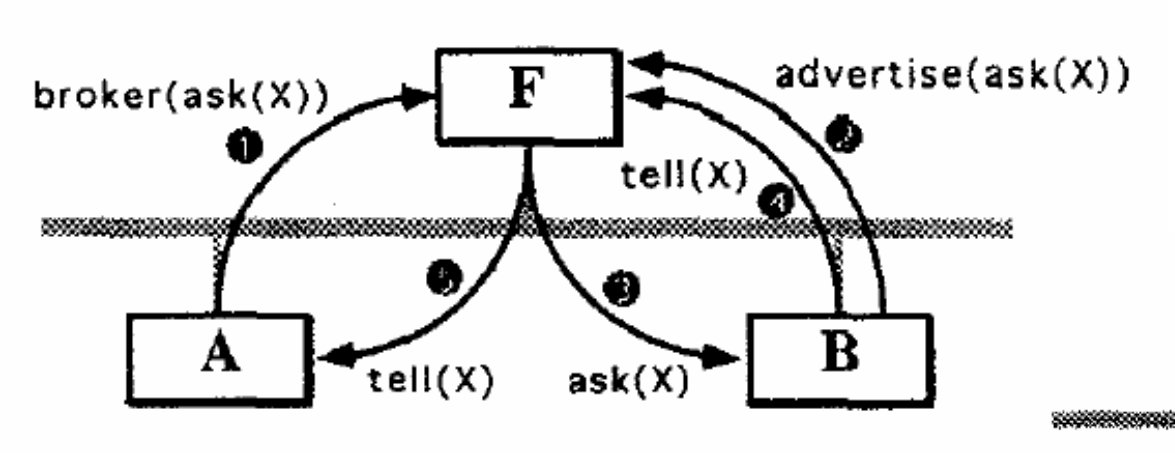
Implementierung

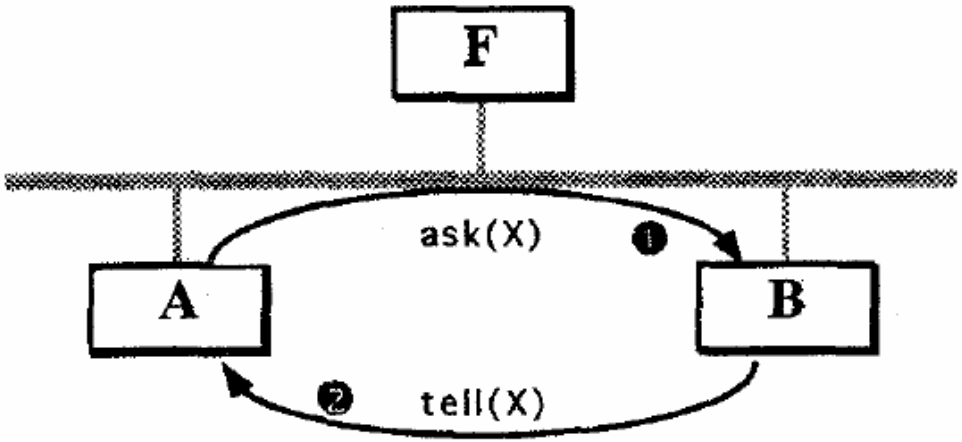
- Kooperation
  - Kooperative Interaktion: Agenten koordinieren ihre Aktionen. Koordination ist durch Programmierer vorgegeben
  - Vertragsbasierte Kooperation: Absichten der Agenten stehen leicht in Konflikt. Koordination durch Marktmechanismen wie Auktionen
  - Verhandlungsbasierte Kooperation: Verhandlungen zwischen Agenten, deren Ressourcenbedarf in Konflikt steht
- Rationalität
  - Alleine: Agent handelt so, dass er seine Ziele erreicht
  - In Gemeinschaft: Agent handelt so, dass der gemeinsame Nutzen größer ist als der gemeinsame Verlust bei einer anderen Aktion

Implementierung

- Kommunikation
  - Agenten müssen Wissen austauschen
  - Agent Communication Languages (ACL)
    - KQML: Sprache zur Äußerung von Kommunikationsakten
      - Auf Sprechakten basiert
      - Mitteilungen haben Typ
      - Definierte Semantik
    - KIF: Sprache zur Repräsentation von Wissen
      - Wie wird der semantische Inhalt der Mitteilung repräsentiert
    - FIPA ACL
      - FIPA Konsortium
  - Ontologien
    - Worüber wird geredet
    - Was sind die Konzepte der Anwendungsdomäne

- (ask-all
  :content „price(IBM, [?price, ?time])“
  :receiver stock-server
  :language standard-prolog
  :ontology NYSE-TICKS)





Beispielsystem

- Zum Ausprobieren:
  JADE (Java Agent DEvelopment Framework)
- http://sharon.cselt.it/projects/jade/
- Java-basiert, FIPA kompatibel, Grafische Oberfläche

Zusammenfassung

Zusammenfassung

1. Schwächen des RPC-Konzepts
   1. Rollen, Transparenz, Fehler, Nebenläufigkeit
2. Koordinationssprachen
   1. Tuplespace
   2. Indirekt, anonym, zeitlich entkoppelt, Mehrparteieninteraktion, nebenläufig, verteilt
   3. XMLSpaces
3. Agenten
   1. Autonomie
   2. ACL

Literatur

- Andrew S. Tanenbaum and Robbert van Renesse. A critique of the remote procedure call paradigm. In Research into Networks and Distributed Applications (EUTECO 88), ed., R. Speth. Elsevier Science Publishers, 1988, pp. 775-783.
- David Gelernter, Nicholas Carriero. Coordination languages and their significance. Communications of the ACM, Volume 35, Issue 2 (February 1992). Pages: 97 - 107
- Eric Freeman, Susanne Hupfer, Ken Arnold. JavaSpaces Principles, Patterns, and Practice. Addison-Wesley, 1999.
- Michael Wooldridge. An Introduction to Multiagent Systems. John Wiley & Sons, 2002.
- Eleni Mangina. Review of Software Products for Multi-Agent Systems. AgentLink (Hrsg.). http://www.agentlink.org/resources/other-pubs.html
- Tim Finin, Richard Fritzson, Don McKay, Robin McEntire. KQML as an agent communication language. Proceedings of the third international conference on Information and knowledge management. 456 – 463. 1994.
- Andrea Omicini, Franco Zambonelli, Matthias Klusch, and Robert Tolksdorf (Editors). Coordination of Internet Agents: Models, Technologies, and Applications. Springer Verlag, 2001. ISBN 3540416137.